Лекция 12. Метод бустинга

1. Введение и история появления бустинга

Большинство людей, причастных к анализу данных, хоть раз слышали про бустинг. Этот алгоритм входит в повседневный "джентльменский набор" моделей, которые стоит попробовать в очередной задаче. Xgboost и вовсе часто ассоциируется со стандартным рецептом для победы в ML соревнованиях, породив мем про "стакать xgboost-ы". А еще бустинг является важной частью большинства поисковых систем, иногда выступая еще и их визитной карточкой. Давайте для общего развития посмотрим, как бустинг появился и развивался.

История появления бустинга

Все началось с вопроса о том, можно ли из большого количества относительно слабых и простых моделей получить одну сильную. Под слабыми моделями мы подразумеваем не просто небольшие и простые модели вроде деревьев решений в противовес более "сильным" моделям, например, нейросетям. В нашем случае слабые модели — это произвольные алгоритмы машинного обучения, точность которых может быть лишь немногим выше случайного угадывания.

Утвердительный математический ответ на этот вопрос нашелся довольно быстро, что само по себе было важным теоретическим результатом (редкость в ML). Однако, потребовалось несколько лет до появления работоспособных алгоритмов и Adaboost. Их общий подход заключался в жадном построении линейной комбинации простых моделей (базовых алгоритмов) путем перевзвешивания входных данных. Каждая последующая модель (как правило, дерево решений) строилась таким образом, чтобы придавать больший вес и предпочтение ранее некорректно предсказанным наблюдениям.

Adaboost работал хорошо, но из-за того, что обоснований работы алгоритма с его надстройками было мало, вокруг них возник полный спектр спекуляций: кто-то считал его сверх-алгоритмом и волшебной пулей, кто-то был скептичен и разделял мнение, что это малоприменимый подход с жесткой переподгонкой (overfitting). Особенно сильно это касалось применимости на данных с мощными выбросами, к которым Adaboost оказался неустойчив. К счастью, когда за дело взялась профессура Стэнфордской кафедры статистики, уже принесшая миру Lasso, Elastic Net и Random Forest, в 1999 году от Jerome Friedman появилось обобщение наработок алгоритмов бустинга — градиентный бустинг, он же Gradient Boosting (Machine), он же GBM. Этой работой Friedman сразу задал статистическую базу для создания многих алгоритмов, предоставив общий подход бустинга как оптимизации в функциональном пространстве.

Постановка ML задачи

Мы будем решать задачу восстановления функции в общем контексте обучения с учителем. У нас будет набор пар признаков x и целевых переменных y,

\large y \approx \hat{f}(x), \\ \large \hat{f}(x) = \underset{f(x)}{\arg\min} \ L(y,f(x))

{(xi,yi)}i=1,…,n на котором мы будем восстанавливать зависимость вида

Бустинг — это техника построения ансамблей, в которой предсказатели построены не независимо, а последовательно

Это техника использует идею о том, что следующая модель будет учится на ошибках предыдущей. Они имеют неравную вероятность появления в последующих моделях, и чаще появятся те, что дают наибольшую ошибку. Предсказатели могут быть выбраны из широкого ассортимента моделей, например, деревья решений, регрессия, классификаторы и т.д. Из-за того, что предсказатели обучаются на ошибках, совершенных предыдущими, требуется меньше времени для того, чтобы добраться до реального ответа. Но мы должны выбирать критерий остановки с осторожностью, иначе это может привести к переобучению. Градиентный бустинг — это пример бустинга.

Алгоритм градиентного бустинга

Градиентный бустинг — это техника машинного обучения для задач классификации и регрессии, которая строит модель предсказания в форме ансамбля слабых предсказывающих моделей, обычно деревьев решений.

Цель любого алгоритма обучения с учителем — определить функцию потерь и минимизировать её. Давайте обратимся к математике градиентного бустинга. Пусть, например, в качестве функции потерь будет среднеквадратичная ошибка (MSE):

$$Loss = MSE = \sum (y_i - y_i^p)^2$$

where, y_i = ith target value, y_i^p = ith prediction, $L(y_i, y_i^p)$ is Loss function

Логика, что стоит за градиентым бустингом, проста, ее можно понять интуитивно, без математического формализма. Предполагается, что читатель знаком с простой линейной регрессией.

Первое предположение линейной регресии, что сумма отклонений = 0, т.е. отклонения должны быть случайно распределены в окрестности нуля.

Теперь давайте думать о отклонениях, как об ошибках, сделанных нашей моделью. Хотя в моделях основанных на деревьях не делается такого предположения, если мы будем размышлять об этом предположении логически (не статистически), мы можем понять, что увидив принцип распределения отклонений, сможем использовать данный паттерн для модели.

Итак, интуиция за алгоритмом градиентного бустинга — итеративно применять паттерны отклонений и улучшать предсказания. Как только мы достигли момента, когда отклонения не имеют никакого паттерна, мы прекращаем достраивать нашу модель (иначе это может привести к переобучению). Алгоритмически, мы минимизируем нашу функцию потерь.

В итоге,

Сначала строим простые модели и анализируем ошибки;

Определяем точки, которые не вписываются в простую модель;

Добавляем модели, которые обрабатывают сложные случаи, которые были выявлены на начальной модели;

Собираем все построенные модели, определяя вес каждого предсказателя.

Шаги построения модели градиентного спуска

Рассмотрим смоделированные данные, как показано на диаграмме рассеивания ниже с 1 входным (x) и 1 выходной (y) переменными.

2. Вычислите погрешности ошибок. Фактическое целевое значение, минус прогнозируемое целевое значение [e1 = y — y_predicted1]

3. Установите новую модель для отклонений в качестве целевой переменной с одинаковыми входными переменными [назовите ее e1_predicted]

4. Добавьте предсказанные отклонения к предыдущим прогнозам

[y_predicted2 = y_predicted1 + e1_predicted]

5. Установите еще одну модель оставшихся отклонений. т.е. [e2 = y — y_predicted2], и повторите шаги с 2 по 5, пока они не начнутся overfitting, или сумма не станет постоянной. Управление overfitting-ом может контролироваться путем постоянной проверки точности на данных для валидации.

тобы помочь понять базовые концепции, вот ссылка с полной реализацией простой модели градиентного бустинга с нуля.

Приведенный код — это неоптимизированная vanilla реализация повышения градиента. Большинство моделей повышения градиента, доступных в библиотеках, хорошо оптимизированы и имеют множество гиперпараметров.

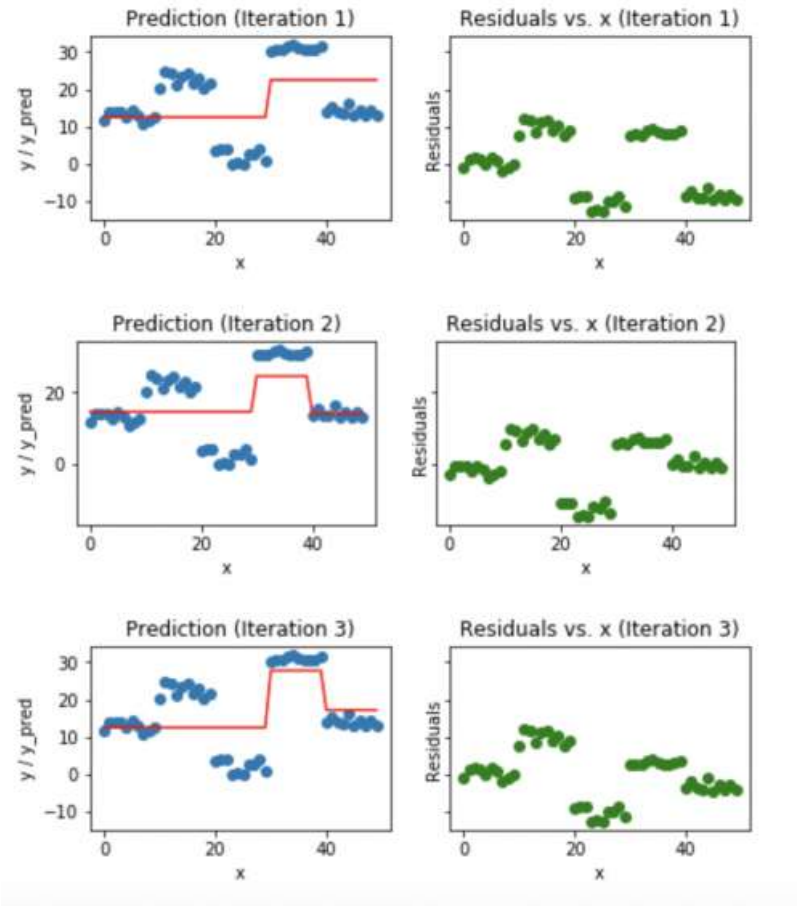
Визуализация работы Gradient Boosting Tree:

Синие точки (слева) отображаются как вход (x) по сравнению с выходом (y);

Красная линия (слева) показывает значения, предсказанные деревом решений;

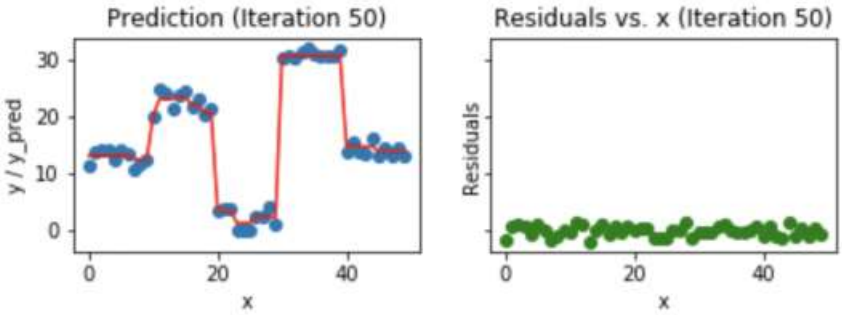
Зеленые точки (справа) показывают остатки по сравнению с вводом (x) для i-й итерации;

Итерация представляет собой последовательное заполнения дерева Gradient Boosting.



Заметим, что после 20-й итерации отклонения распределены случайным образом (здесь не говорим о случайной норме) около 0, и наши прогнозы очень близки к истинным значениям (итерации называются n_estimators в реализации sklearn). Возможно, это хороший момент для остановки, или наша модель начнет переобучаться.

Посмотрим, как выглядит наша модель после 50-й итерации.



Мы видим, что даже после 50-й итерации отклонения по сравнению с графиком x похожи на то, что мы видим на 20-й итерации. Но модель становится все более сложной, и предсказания перерабатывают данные обучения и пытаются изучить каждый учебный материал. Таким образом, было бы лучше остановиться на 20-й итерации.